CSE-276C HW2

Zhenyu Wu | PID: A69030822

November 1, 2024

1 Question 1

1.1 Question 1.1

The graph of ground truth trajectory is shown by Fig.1.

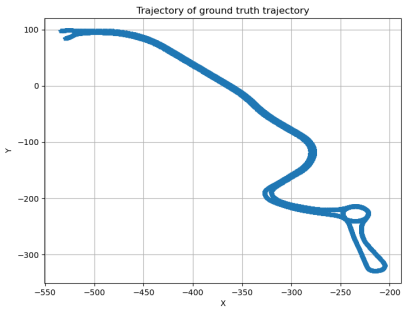
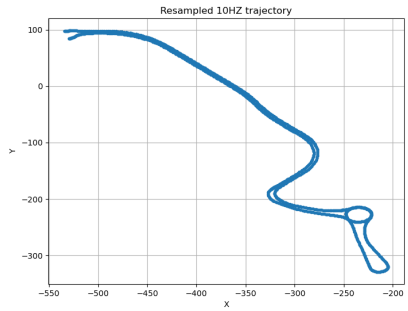


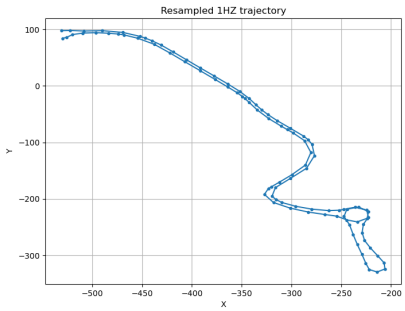
Figure 1: Ground True Trajectory

1.2 Question 1.2

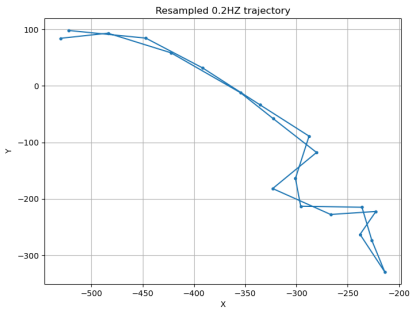
To begin with, I plotted the trajectory of downsampled data, which are shown below.



DownSample 10Hz Trajectory



DownSample 1Hz Trajectory



DownSample 0.2Hz Trajectory

My general thoughts to solve this question are: 1. Down sample data, pair each x and y with a timestamp to simulate the down sample frequency. 2.Do the interpolation.

Actually, my initial attempt was to set unsampled data y as NaN and keep the x, and use interpolation on x to get unsampled y. However, this didn't work. For the 10Hz and 1HZ, this method gives a not bad result, I assumed that it was because the data downsampled to 10Hz and 1Hz were not too sparse (Their

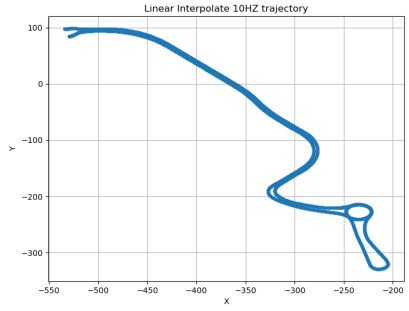
trajectory didn't lose too much details). However, for the 2Hz, this method generates a really wrong results. I also noticed that in documentations of many tools such as numpy, scipy, they all pointed out that if x is not monotonically increasing, the result would be wrong and it is hard to determine the trend. And numpy and some packages would do sorting before interpolation.
Then, to make a monotonically increasing "x", I found out that I can treat x, y as separate function value of timestamp Δt. For each frequency(10, 1, 0.2), I applied interpolation twice, one on $(\Delta t, x)$ and another on $(\Delta t, y)$. Then I put x,y together to plot the trajectory $xvs.y$ and calculate the error between interpolated (x, y) and unsampled ground truth (x^*, y^*)

1.3 Question 1.2.a

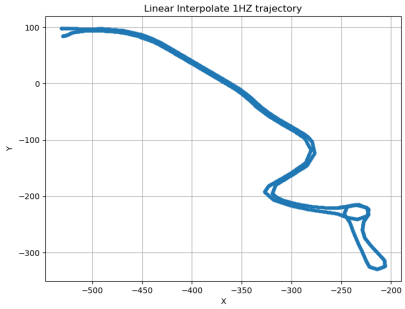
For the linear interpolation, I tried to implement it by myself at the first place. However, I was afraid I would make a mistake in implementation just like the normalization in HW1. Then I turned to use numpy to do linear interpolation on (t, x) and (t, y). The numpy algorithm basically do the sorting first to ensure the monotonically increasing "x", then it calculate

$$y = y_1 + \frac{(x - x_1)(y_2 - y_1)}{x_2 - x_1}$$

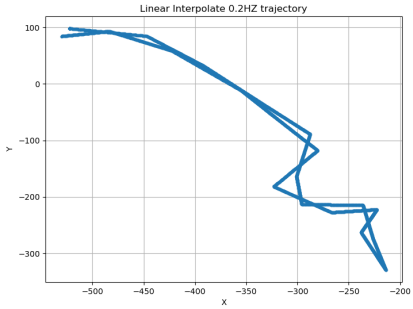
The trajectory I have for linear interpolation are shown below



Linear Interpolate 10Hz Trajectory



Linear Interpolate 1Hz Trajectory



Linear Interpolate 0.2Hz Trajectory

The total error(Euclidean distance) of all unsmapled ground truth (x^*, y^*) and interpolated (x, y) are:

10HZ Cummulative Error of Linear Interpolation: 45.77082837475039

1HZ Cummulative Error of Linear Interpolation: 1530.6420811644919

0.2HZ Cummulative Error of Linear Interpolation: 24109.30874076987

For 0.2Hz, it lost too much details, and linear interpolation was just "connecting adjacent points" which gave too much straight edges in the trajectory.

1.4 Question 1.2.b

For the quadratic interpolation, I applied the scipy interpolate library. It will take three adjacent points $(t_0, x_0), (t_1, x_1), (t_2, x_2)$ to build a second order polynomial and find the coefficient of it.

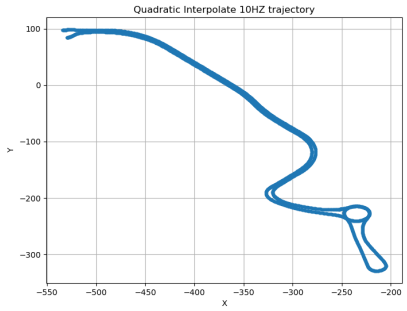
$$y = y_0 \times L_0(x) + y_1 \times L_1(x) + y_2 \times L_2(x) \tag{1}$$

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} \tag{2}$$

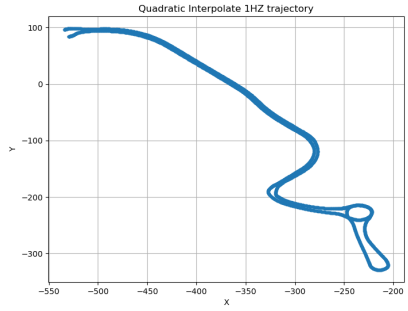
$$L_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} \tag{3}$$

$$L_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} \tag{4}$$

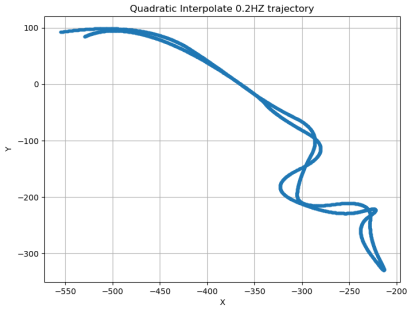
A separate quadratic polynomial is created for each segment between adjacent data points, and each polynomial segment smoothly connects one point to the next.
The trajectory I have for Quadratic interpolation are shown below



Quadratic Interpolate 10Hz Trajectory



Quadratic Interpolate 1Hz Trajectory



Quadratic Interpolate 0.2Hz Trajectory

The total error(Euclidean distance) of all unsmapled ground truth (x^*, y^*) and interpolated (x, y) are:

10HZ Cummulative Error of Quadratic Interpolation: 45.37993363557641

1HZ Cummulative Error of Quadratic Interpolation: 520.8404259101981

0.2HZ Cummulative Error of Quadratic Interpolation: 19260.701839123296

There is only minor improvement for 10HZ data, I think it is because for 10Hz, we still have enough data without lossing important information of trajectory.

But it has a quite big improvement for 1Hz and 0.2Hz. Since quaradic interpolation can do curve fitting. Unlike linear interpolation, it will not make straight edges. Instead, it will generate a curve between adjacent points which could better reflect the curvature of the original trajectory, particularly during turns or more complex movements like a circular turn at lower right corner of the trajectory. This smoother fit reduces the sharp angles that linear interpolation would introduce.

1.5 Question 1.2.c

For the cubic spline, I also applied the scipy interpolate library, it will find the coefficient of

$$A = \frac{x_{i+1} - x}{x_{i+1} - x_i}$$

$$B = \frac{x - x_i}{x_{i+1} - x_i}$$

$$C = \frac{(x_{i+1} - x)^2(x - x_i)}{6(x_{i+1} - x_i)}$$

$$D = \frac{(x - x_i)^2(x_{i+1} - x)}{6(x_{i+1} - x_i)}$$

$$y = Ay_i + By_{i+1} + Cy_i'' + Dy_{i+1}''$$

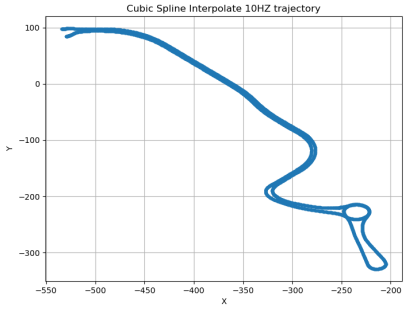
I also noticed that scipy provides some boundary condition to input, which are natural spline (second derivative at the endpoints is set to zero), clamped spline (first derivative at the endpoints is specified) and Not-a-knot spline (default condition, third derivative at the second and second-last points is continuous). I tried all three of them. (For the clamped spline, I used the slope at end points to estimate first derivative). It turns out that the natural spline where second derivative at the endpoints is set to zero had the lowest total error, which is

10HZ Cummulative Error of Cubic Interpolation: 45.92407605233177

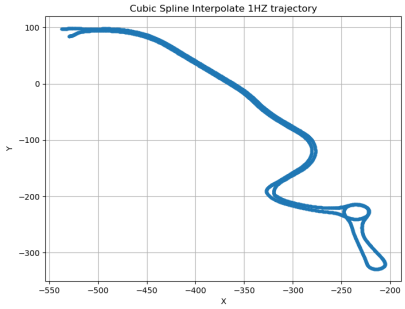
1HZ Cummulative Error of Cubic Interpolation: 498.7414404415983

0.2HZ Cummulative Error of Cubic Interpolation: 18439.79645738146

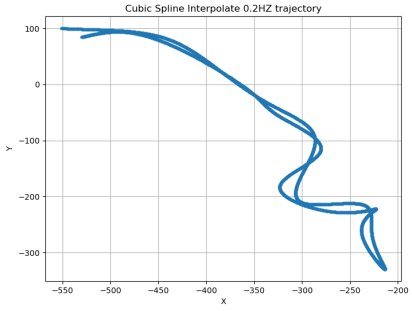
The trajectory I have for Cubic interpolation are shown below



Cubic Spline Interpolate 10Hz Trajectory



Cubic Spline Interpolate 1Hz Trajectory



Cubic Spline Interpolate 0.2Hz Trajectory

It has some improvement than quaradic interpolation. I assume it is because the cublic spline maintain continuity up to the second derivative which can better model the changes in both slope and curvature.

1.6 Question 1.3

Like I just mentioned, under linear interpolation, the low sample rate is most detrimental, since we have lost many information about the slope and curvature of the trajectory. Linear interpolation would generate many straight edges which will not fit data well. And cubic spline has the lowest total error, which might because it could can better model the changes in both slope and curvature.

1.7 Question 1.4

Linear interpolation connects each pair of adjacent data points with a straight line, I think it would have better performance when data points are closely spaced and the underlying function is approximately linear between points, but it performs poorly when the true function has significant curvature between data points. It might be a good choice if the data is discontinuous or has spikes(sharp changes)
Quadratic interpolation fits a second-degree polynomial to a set of data points which applies adjacent three points to construct a segment, if the data has reasonable curvature it might be a good choice, but it might introduce errors if the function significantly deviates within an interval.
Cubic interpolation constructs cubic polynomials between each pair of data points and ensures continuity in the first and second derivatives across all intervals, creating a smooth curve. I think it can outperform the other two when data has higher change in slope and curvature or continuous derivatives.

2 Question 2

For $f(x) = cos(x)$ over interval $[0, \pi]$, given the Weierstrass Approximation Theorem and we need to ensure 6 decimal digit accuracy, we have:

$$\|f(x) - p(x)\|_\infty = \max_{x \in [0,\pi]} |f(x) - p(x)| < 10^{-6}$$

2.1 Question 2.a

Based on the error bound I listed above, my first thought was to find a way to express the maximum error $E_{max}(x) = \|f(x) - p(x)\|_\infty < 10^{-6}$. Then I did some research about maximum error bound of polynomial interpolation, and I found two posts at `https://math.stackexchange.com/questions/1151001/lagrange-interpolating-polynomials-error-bound` and `https://math.stackexchange.com/questions/3989113/show-that-the-maximum-error-in-a-linear-interpolation-is-given-by-fra` `E2%88%92x`. They stated that for a polynomial interpolation on $f(x)$ which is $n + 1$ times differentiable and continuous over closed interval $[a, b]$, the error bound can be expressed as:

$$f(x) - P(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0) \cdots (x - x_n)$$

For linear interpolation, we have $n = 1$, therefore,

$$f(x) - P(x) = \frac{f^{''}(\xi(x))}{2}(x - x_0)(x - x_1)$$

and the second post provides a complete process of deriving the maximum error bound of the right hand side for linear interpolation, which is just solve $\max|(x - x_0)(x - x_1)|$

$$E_{max}(x) = \frac{(x_1 - x_0)^2}{8} \max_{x \in [x_0, x_1]} |f^{''}(x)|$$

Since we are applying uniform spacing between sampled points, we need to ensure the 6 digit accuracy of linear interpolation between x_i and x_{i+1} for every x_i and x_{i+1} sampled from $[0, \pi]$, which means

$$E_{max}(x) = \frac{(x_{i+1} - x_i)^2}{8} \max_{x \in [x_i, x_{i+1}]} |f^{''}(x)| = 10^{-6}$$

For $f(x) = cos(x)$ over $[0, \pi]$, the maximum $f^{''}(x) = 1$, therefore we have:

$$\frac{(x_{i+1} - x_i)^2}{8} = 10^{-6}$$

$$(x_{i+1} - x_i) = 2.828427125 \times 10^{-3}$$

where $(x_{i+1} - x_i)$ is the table spacing.

2.2 Question 2.b

Similar to the approach I mentioned in 2.a, I need to find the maximum error bound for quadratic interpolation as

$$E_{max}(x) = \frac{\max|(x - x_i)(x - (x_i + h))(x - (x_i + 2h))|}{3!} \max|f^{'''}(x)|$$

where h is the uniform space between each points. For the $f^{'''}(x) = sin(x)$, then $\max|f^{'''}(x)| = 1$
To drive $\max|(x - x_i)(x - (x_i + h))(x - (x_i + 2h))|$, let $s = x - x_i$, we have

$$F(s) = s(s - h)(s - 2h) = s^3 - 3s^2h + 2sh^2$$

$$F'(s) = s^2 - 3sh + 2h^2$$

Then I solve for $F'(s) = 0$

$$s^2 - 3sh + 2h^2 = 0$$

$$s_1 = h - \frac{\sqrt{3}}{3}h, \quad s_2 = h + \frac{\sqrt{3}}{3}h$$

Then I substitute s back to $F(s)$ I have

$$\max|F(s)| = \frac{\sqrt{3}}{9}h^3 = \max|(x - x_i)(x - (x_i + h))(x - (x_i + 2h))|$$

Put together I have

$$E_{max}(x) = \frac{\frac{\sqrt{3}}{9}h^3}{6} = 10^{-6}$$
$$h = 0.0314734519$$

where h is the uniform space.

2.3 Question 2.c

To find the number of entries, I just use the $\frac{\pi}{h}$ where h is the uniform space for each interpolation.
For Linear Interpolation:

$$n = 1110.720734 \approx 1111$$

For Quadratic Interpolation:

$$n = 99.81722575 \approx 100$$